Introduction : algorithmically undecidable problems

There are inherent limitations on what [mathematical] problems can be solved using computers. Even with the idealization that the amount of time & working space available to carry out a computation is unlimited, there exist problems that are computationally unsolvable.

Three famous examples sketched in this lecture :

- Hilbert's Entscheidungsproblem
- The Halting Problem
- Hilbert's 10th Problem

Hilbert's Entscheidungsproblem

Is there an algorithm [aka "effective procedure"] **which when fed any statement in the formal language of first order arithmetic, determines in a finite number of operations whether the statement is provable from Peano's axioms for arithmetic using the usual rules of classical logic ?**

Posed by Hilbert at the 1928 International Congress of Mathematicians, and in fact the problem was stated in a more ambitious form, with a more powerful formal system in place of first order arithmetic.

The algorithm Hilbert's Entscheidungsproblem asks for would be a rather useful thing to have ! E.g. we could run it on the statement

$$\forall k > 1.\ \exists p, q.\ 2k = p + q\ \&\ \text{prime}(p)\ \&\ \text{prime}(q)$$

to find out whether Goldbach's Conjecture has a proof; etc., etc.

Hilbert believed that such an algorithm could be found. In 1930 he wrote

"In an effort to give an example of an unsolvable problem, the philosopher Comte once said that science would never succeed in ascertaining the secret of the chemical composition of the bodies of the universe. A few years later this problem was solved ... The true reason, according to my thinking, why Comte could not find an unsolvable problem lies in the fact that there is no such thing as an unsolvable problem."

[quoted from C. Reid's biography of Hilbert]

A few years later Hilbert was proved wrong, by Church and Turing's work of 1935-36.

'Entscheidungsproblem' means 'decision problem'.

General form of a **decision problem** specified by:

- set S whose elements are finite datastructures of some kind (eg: formulas in formal system for first order arithmetic)
  - infinite, if the problem is to be non-trivial
- property P of elements of S (eg: property of a formula that it has a proof)

The problem is then: find an algorithm A which
- always terminates with result 0 or 1 when fed an element $s \in S$, and
- yields result 1 when fed s if & only if s has prop. P

write $A(s) = 1$ to indicate this

Algorithms, or "effective procedures"

No precise definition at time Hilbert posed the Entscheidungsproblem, just examples ...

Common features of the examples:

- finite description of the procedure in terms of "elementary" operations
- deterministic (i.e. "next step" is uniquely determined, if there is one)
- procedure may not terminate on some input data, but can recognize when it does & what the result is

Examples of algorithms abound in the history of mathematics, eg:

- procedure for multiplying numbers in decimal notation
- procedure for extracting square roots to any desired accuracy
- procedure for finding highest common factors (Euclid's Algorithm)

etc., etc.

In 1935/36 Turing and Church gave independent, negative solutions of Hilbert's Entscheidungsproblem. The essential first step was to formulate a precise, mathematical definition of the notion of 'algorithm'.

Turing's formulation (in terms of what are now called Turing machines — of which more later) appeared more general/fundamental than Church's formulation (in terms of his lambda calculus), but Turing proved that both formulations described the same class of functions from [numerical] inputs to [numerical] outputs.

(Turing machines prefigured, and partly stimulated, the early development of digital computing. Lambda calculus partly inspired the development of 'functional' programming languages.)

Next step of Turing/Church solution of the Entscheidungsproblem: with a precise definition of 'algorithm', one can regard algorithms as data on which algorithms can act. Hence one can consider...

The Halting Problem

= the decision problem with

S = set of all pairs (A, D) where A is an algorithm and D is a datum on which it is designed to operate

P = property of such pairs given by:

"algorithm A when applied to datum D eventually produces a result (i.e. eventually halts)"

write $A(D)\downarrow$ to indicate this

Turing and Church showed that the Halting Problem is undecidable, i.e. there is no algorithm H such that for all $(A, D) \in S$

$$H(A,D) = \begin{cases} 1 & \text{if } A(D)\downarrow \\ 0 & \text{otherwise} \end{cases}$$

Sketch of the proof

If there were such an H, we could use it to define an algorithm C:

"input A, compute $H(A,A)$ and if it is equal to 0 then return value 1 & halt, otherwise loop forever."

So for all A, $C(A)\downarrow \iff H(A,A) = 0$ (since H total)

and for all A, $H(A,A) = 0 \iff A(A)\not\downarrow$ (by def^n of H)

Hence for all A, $C(A)\downarrow \iff A(A)\not\downarrow$

Taking A to be C itself, we get $C(C)\downarrow \iff C(C)\not\downarrow$ contradiction!

The final step in the Turing/Church proof of the undecidability of Hilbert's Entscheidungsproblem was to show that the problem can be reduced to the Halting Problem, by constructing an algorithm for encoding instances (A, D) of the Halting Problem set as arithmetic statements $\Phi_{A,D}$ such that

$\Phi_{A,D}$ is provable $\iff$ $A(D)\downarrow$

Hence any algorithm for deciding provability of arithmetic statements could be used to solve the Halting Problem – so no such can exist.

With hindsight, a positive solution to the Entscheidungsproblem would be too good to be true. However, the algorithmic unsolvability of some decision problems is much more surprising. A famous example of this is...

Hilbert's 10th Problem

Give an algorithm which, when started with any Diophantine equation, determines in a finite number of operations whether there are natural numbers satisfying the equation.

One of a number of important open problems listed by Hilbert at the International Congress of Mathematicians in 1900.

Diophantine equations

$$p(x_1,\ldots,x_n) = q(x_1,\ldots,x_n)$$

p, q polynomials in $x_1,\ldots,x_n$ with coefficients from $\mathbb{N} = \{0,1,2,3,\ldots\}$.

Named after Diophantus of Alexandria (c. 250 AD)

E.g. "Find three whole numbers (x,y,z) such that the product of any two added to the third is a square" [Diophantus' Arithmetica, Book III, Problem 7]

i.e. find $x,y,z \in \mathbb{N}$ for which there exists $u,v,w \in \mathbb{N}$ with $(xy+z-u^2)^2+(yz+x-v^2)^2+(zx+y-w^2)^2 = 0$

i.e. with $(x^2y^2+y^2z^2+z^2x^2+\ldots) = (u^2xy+v^2yz+w^2zx+\ldots)$

[One solution: $(x,y,z)=(1,4,12)$, with $(u,v,w)=(4,7,4)$.]

Hilbert's 10th Problem was eventually shown to be reducible to the Halting Problem and hence algorithmically undecidable only in 1970 by the combined efforts of Y. Matijasevič, J. Robinson, M. Davis and H. Putnam. The original proof used Turing machines and was quite complicated. Later J.P. Jones & Y. Matijasevič (Jour. Symb. Logic 49(1984) 818-829) simplified a large part of the proof by using a formulation of algorithm (equivalent to Turing machines or λ-calculus) in terms of register machines – a formulation invented by Minsky & Lambek in the 1960s.

We will use register machines to develop the ideas sketched in this lecture and make them precise. We will return to Turing and Church's formulation of the notion of algorithm later in the course.